UC Workbench - A Tool for Writing Use Cases and Generating Mockups

Jerzy Nawrocki and Łukasz Olek

Poznań University of Technology, Institute of Computing Science,
ul. Piotrowo 3A, 60-965 Poznań, Poland
{Jerzy.Nawrocki, Lukasz.Olek}@cs.put.poznan.pl

Abstract. Agile methodologies are based on effective communication with the customer. The ideal case is XP's on-site customer. Unfortunately, in practice customer representatives are too busy to work with the development team all the time. Moreover, frequently there are many of them and each representative has only partial domain knowledge. To cope with this we introduced to our projects a proxy-customer role resembling RUP's Analyst and we equipped him with a tool, called UC Workbench, that supports the communication with the customer representatives and the developers. Analyst collects user stories from customer representatives and 'translates' them into use cases. UC Workbench contains among other things a use-case editor and a generator of mockups (a mockup generated by UC Workbench animates use-cases and illustrates them with screen designs).

1 Introduction

Many agile methodologies use informal stories for requirements description [3]. Perhaps the most popular are XP's user stories [2] [5]. An interesting alternative to them are use cases invented by Ivar Jacobson [7] and incorporated into Rational Unified Process. They provide "*a semiformal framework for structuring the stories*" [1]. Although they are more formal than user stories, they are not contradictory with the agile approach to software development. There are some agile methodologies that use written requirements or permit them (e.g. the Crystal methodologies [4], DSDM [14] or Scrum [13]). For people applying those methodologies, use cases can be very useful. As Craig Larman put it: "*when written functional requirements are needed, consider use cases*" [9].

Use-cases engineering comprises *editing* the use cases (e.g. inserting a step can require re-numbering all the subsequent steps and extensions associated with them - that could be done automatically) and *generating mockups* that would animate the use cases (that would support customer-developers communication), and *preparing effort calculators* based on Use-Case Points [11] and adjusted to the current set of use cases (that could be a valuable tool supporting XP's Planning Game or backlog's effort estimation in Scrum). Unfortunately, there is no tool offering that range of functionality. The only thing we have found was a use-case editor, called CaseComplete, offered by Serlio Software [16].

The aim of the paper is to describe our approach to use-case engineering. It is based on a tool called *UC Workbench* (Use Case Workbench) developed at the Poznan University of Technology. It is a use-case editor combined with a mockups generator and an effort-calculators generator. It is important for an agile team that after changing some use cases a new mockup and updated effort calculators are obtained 'at the press of a button'. UC Workbench has been successfully used at the university's Software Development Studio and in a commercial project run by PB Polsoft, a medium size software company with headquarters in Poznan. In the paper we focus on the language for use-cases description (Sec. 2.), and generation of mockups (Sec. 3.). Other functionality provided by the tool includes automatic inspections and support for effort estimation.

2 FUSE: A Language for Use-Cases Description

Use cases collected by an analyst are edited with the help of UC Editor, a part of UC Workbench. The editor uses FUSE (Formal USE cases) language. It is a simple language formalizing structure of use-cases description to allow generating of mockups and effort calculators (actor descriptions and steps within the use cases are expressed in a natural language).

FUSE is based on use-case patterns collected by Adolph and his colleagues [1]. Use cases are accompanied with actor descriptions (that is adviced by the Clear-CastOfCharacters pattern). FUSE allows for two forms of use cases (the MultipleForms pattern): *casual* and *formal*. The former has no steps, just plain text and resembles XP's user stories. The latter corresponds to the ScenarioPlusFragments pattern: the description consists of a main scenario split into a number of steps and extensions (that form is very popular - see e.g. [6]). When applying the breadth-before-depth strategy and spiral development, first the casual form is used and at the next cycle some use cases are refined and written down using the formal form.

To make formal descriptions of use cases more precise and readable we have decided to introduce the Either-Or construct to FUSE. Moreover, for the sake of readability FUSE allows nested steps that are especially helpful when combined with Either-Or.

2.1 Either-Or

Sometimes one needs *nondeterministic choice* between alternative steps. Assume, for instance, that someone has to describe how to buy books in an Internet-based bookstore. A customer can either *add a book to the cart* or *remove one from it.*

Putting those two steps in a sequence

```
1. Customer adds a book to the cart.
2. Customer removes a book from the cart.
```

is not correct, as it suggests that one has always to remove a book adding one to the cart. A remedy could be to add an extension like the following one

```
2a. Customer does not want to remove a book from the cart.
    2a1. Customer skips the step.
```

to each of the above steps. Unfortunately, that would clutter the description and other really important extensions would be less visible.

To solve the problem we have decided to introduce the Either-Or construct to the formal form of use cases. Using it one could describe customer's options in the following way:

```
1. Either: Customer adds a book to the cart.
2. Or:     Customer removes a book from the cart.
```

One can argue that the Either-Or construct can be difficult for some end-users to understand. In the case of UC Workbench it should not be a problem, as the use cases are accompanied with an automatically generated mockup which visualizes the control flow by animating of use cases.

2.2 Nested steps

Sometimes a step can be decomposed into 2 or 3 other steps. Then it can be convenient to have the "substeps" shown directly in the upper level use case (according to the LeveledSteps pattern a scenario should contain from 3 to 9 steps). For instance, assume that adding a book to the cart consists of the following "substeps":

```
1. Customer selects a book.
2. System shows new value of the cart.
```

Then buying books could be described in FUSE in the following way (use-case header and extensions have been omitted):

```
1. Either: Customer adds a book to the cart.
           1.1. Customer selects a book.
           1.2. System shows new value of the cart.
2. Or:     Customer removes a book from the cart.
```

Again, end-user will be supported with a mockup helping him to understand the control flow, but if the analyst thinks it is not enough she can always choose not to use new constructs.

An example of use case written in FUSE is presented below:

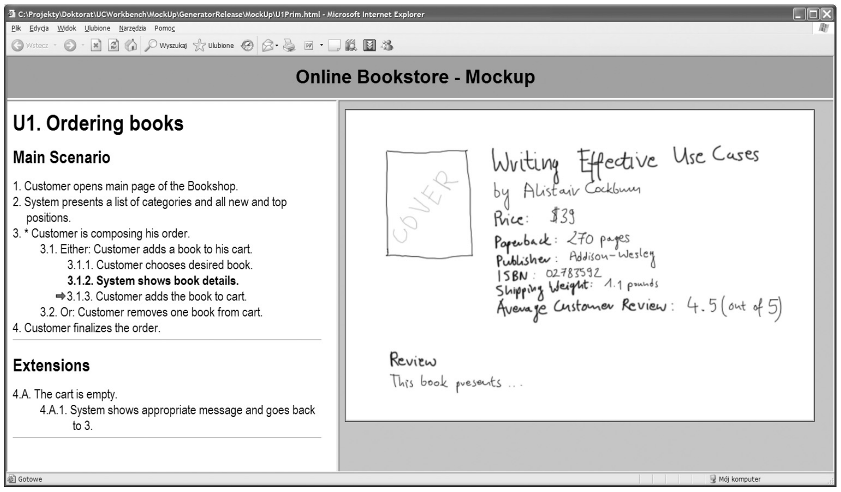
```
Main scenario:
  1. Customer opens main page of a Bookshop.
  2. System presents a list of categories and all new positions.
  3. Customer is composing his order:
    3.1. Either: Customer adds a book to his cart:
      3.1.1. Customer chooses a desired book.
      3.1.2. System shows the book details.
      3.1.3. Customer adds the book to cart.
    3.2. Or: Customer removes a book from the cart.
  4. Customer finalizes the order.
Extensions:
  4.A. The cart is empty.
    4.A.1. System shows appropriate message and returns to step 3.
```

3 Generating of Mockups

There are two kinds of prototypes [10]: throwaway prototypes (mockups) and evolutionary ones. The latter are core of every agile methodology. The former could be used to support customer-developers communication about requirements but their development had to be very cheap and very fast.

The mockups generated by UC Workbench are simple but effective. They focus on presenting functionality. They combine use cases (i.e. behavioural description) with screen designs associated with them (that complies with the Adornments pattern [1]). A generated mockup is based on a web browser and it consists of two frames (see Fig. 1):

- the *scenario window* presents the currently animated use cases (it is the left frame in Fig. 1) and the current step is shown in bold;
- the *screen window* shows the screen design associated with the current step (it is the right frame in Fig. 1).


C:\Projekty\Doktorat\UCWorkbench\MockUp\GeneratorRelease\MockUp\U1Prim.html - Microsoft Internet Explorer
Plik Edycja Widok Ulubione Narzędzia Pomoc
Wstecz Wyszukaj Ulubione
Gotowe
Mój komputer

Online Bookstore - Mockup

U1. Ordering books

Main Scenario

1. Customer opens main page of the Bookshop.
2. System presents a list of categories and all new and top positions.
3. * Customer is composing his order.
    3.1. Either: Customer adds a book to his cart.
        3.1.1. Customer chooses desired book.
        3.1.2. System shows book details.
        ⇒3.1.3. Customer adds the book to cart.
    3.2. Or: Customer removes one book from cart.
4. Customer finalizes the order.

Extensions

4.A. The cart is empty.
    4.A.1. System shows appropriate message and goes back to 3.

COVER

Writing Effective Use Cases
by Alistair Cockburn
Price: $39
Paperback: 270 pages
Publisher: Addison-Wesley
ISBN: 02783592
Shipping Weight: 1.1 pounds
Average Customer Review: 4.5 (out of 5)

Review
This book presents ...

Fig. 1. A Mockup Screen

To generate a mockup the analyst has to associate with use-case steps names of files containing screen designs. The fidelity levels of screen designs are up to the analyst (an interesting discussion about fidelity levels can be found in [12], [15], [8]). The screen design shown in Fig. 1 is at the low fidelity level and it has been created with a tablet connected to PC. After decorating 'difficult' use-cases with screen designs one can generate a mockup "at the press of a button". Using UC Workbench one can produce a mockup (i.e. re-write use cases and adorn them with screen designs) within a few hours — that goes well with agile methodologies and can really support customer-developers communication, especially if there is a danger that the requirements are ambiguous or contradictory.

4 Conclusions

UC Workbench presented in the paper supports editing use cases (we were supprized that Rational Requisite Pro much more supports 'traditional' requirements than use cases) and generates mockups that animate use cases. What is important for agile developers a mockup can be obtained automatically, so it is always consistent with changing requirements. *UC Workbench* supports also automatic inspections, effort estimation based on Use Case Points [11] and generation of the software requirements document from a set of use cases.

Acknowledgements

We would like to thank Grzegorz Leopold and Piotr Godek from PB Polsoft – their bravity allowed us to get a feedback from industry and helped us to improve the tool. This work has been financially supported by the State Committee for Scientific Research as a research grant 4 T11F 001 23 (years 2002-2005).

References

1. Adolph S., Bramble P., Cockburn A., Pols A.: Patterns for Effective Use Cases. Addison-Wesley (2002)
2. Beck, K.: Extreme Programming Explained. Embrace Change. Addison-Wesley, Boston, 2000
3. Boehm, B., Turner, R: Balancing Agility and Discipline. A Guide for the Perplexed. Addison-Wesley, Boston, 2004
4. Cockburn, A.: Agile Software Development. Addison-Wesley, Boston, 2002.
5. Cohn, M.: User Stories Applied. Addison-Wesley, Boston, 2004.
6. Fowler, M., Scott, K.: UML Distilled. Addison-Wesley, Boston, 2000.
7. Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Reading MA, 1992.
8. Landay J.A.: SILK: Sketching Interfaces Like Crazy IEEE Computer-Human Interaction (April 13-18, 1996)
9. Larman, C.: Agile And Iterative Development. A Manager's Guide. Addison-Wesley, Boston, 2004.
10. Pressman, R.: Software Engineering. A Practitioner's Approach. McGrow-Hill, New York, 1997.
11. Ribu K.: Estimating Object-Oriented Software Projects with Use Cases Master of Science Thesis, University of Oslo 2001
12. Rittig M.: Prototyping for Tiny Fingers. Communications of the ACM, April 1994/Vol. 37, No. 4 p. 21-27
13. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press, Redmond, 2004.
14. Stapleton, J.: DSDM. Business Focused Development. Addison-Wesley, London, 2003.
15. Walker M., Takayama L., Landay J.A.: High-fidelity or Low-fidelity, Paper or Computer? Choosing Attributes When Testing Web Prototypes. Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting: HFES2002. pp. 661-665
16. http://www.serliosoft.com/casecomplete/